Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №5

Эмуляция АЛУ. Операция умножения

чисел с плавающей точкой

Студент: гр. 853504

Яскевич Никита Николаевич

Руководитель: старший преподаватель

Шиманский В.В.

Минск 2020

СОДЕРЖАНИЕ

1. Введение
2. Постановка задачи
3. Программная реализация
4. Выводы

Литература

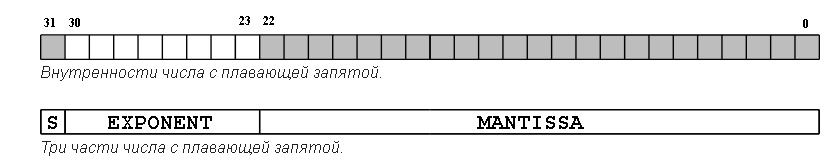
Приложение 1 – Текст программы

1.Введение

### 1.1. **Числа с плавающей запятой**

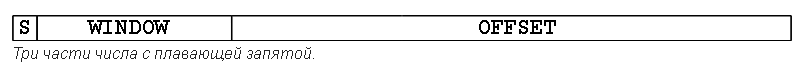
Важно понимать принципы её работы, чтобы полностью осознать её полезность при программировании 3D-движка. В языке C значения с плавающей запятой — это 32-битные контейнеры, соответствующие стандарту IEEE 754. Они предназначены для хранения и выполнения операций над аппроксимациями вещественных чисел. Пока я видел только такое их объяснение. 32 бита разделены на три части:

* S (1 бит) для хранения знака
* E (8 бит) для экспоненты
* M (23 бита) для мантиссы



### **Другой способ представления**

К счастью, их можно объяснить иначе. Воспринимайте экспоненту как окно (Window) или интервал между двумя соседними целыми степенями двойки. Мантиссу воспринимайте как смещение (Offset) в этом окне.



Окно сообщает нам, между какими двумя последовательными степенями двойки будет число: [0,1], [1,2], [2,4], [4,8] и так далее (вплоть до [2^127,2^128]. Смещение разделяет окно на 223=8388608 сегментов. С помощью окна и смещения можно аппроксимировать число. Окно — это отличный механизм защиты от выхода за границы. Достигнув максимума в окне (например, в [2,4]), можно «переплыть» вправо и представить число в пределах следующего окна (например, [4,8]). Ценой этого будет только небольшое снижение точности, потому что окно становится в два раза больше.

1.2. Нормальная и нормализованная формы

*Нормальной формой* числа с плавающей запятой называется такая форма, в которой мантисса (без учёта знака) находится на полуинтервале [0 1), то есть 0 ⩽ a < 1.

Такая форма записи имеет недостаток: некоторые числа записываются неоднозначно (например, 0,0001 можно записать как 0,000001⋅102, 0,00001⋅101, 0,0001⋅100, 0,001⋅10−1, 0,01⋅10−2 и так далее), поэтому распространена (особенно в информатике) также другая форма записи — *нормализованная*, в которой мантисса десятичного числа принимает значения от 1 (включительно) до 10 (исключительно), то есть 1 ⩽ a < 10 (аналогично мантисса двоичного числа принимает значения от 1 до 2). В такой форме любое число (кроме 0) записывается единственным образом. Недостаток заключается в том, что в таком виде невозможно представить 0, поэтому представление чисел в информатике предусматривает специальный признак (бит) для числа 0.

Старший разряд (целая часть числа) мантиссы двоичного числа (кроме 0) в нормализованном виде равен 1 (так называемая *неявная* единица), поэтому при записи мантиссы числа в ЭВМ старший разряд можно не записывать, что и используется в стандарте IEEE 754. В позиционных системах счисления с основанием большим, чем 2 (в [троичной](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), четверичной и др.), этого свойства нет

**1.3. Алгоритм умножения.**

Если заданы A = a \* 2m и B = b \* 2m в нормальной форме, то их произведение составит :

AB = ab \* 2m1+m2

**Операция умножения** состоит из следующих **этапов**:

1) Определение знака произведения путем сложения по mod 2 знаковых цифр мантисс сомножителей.

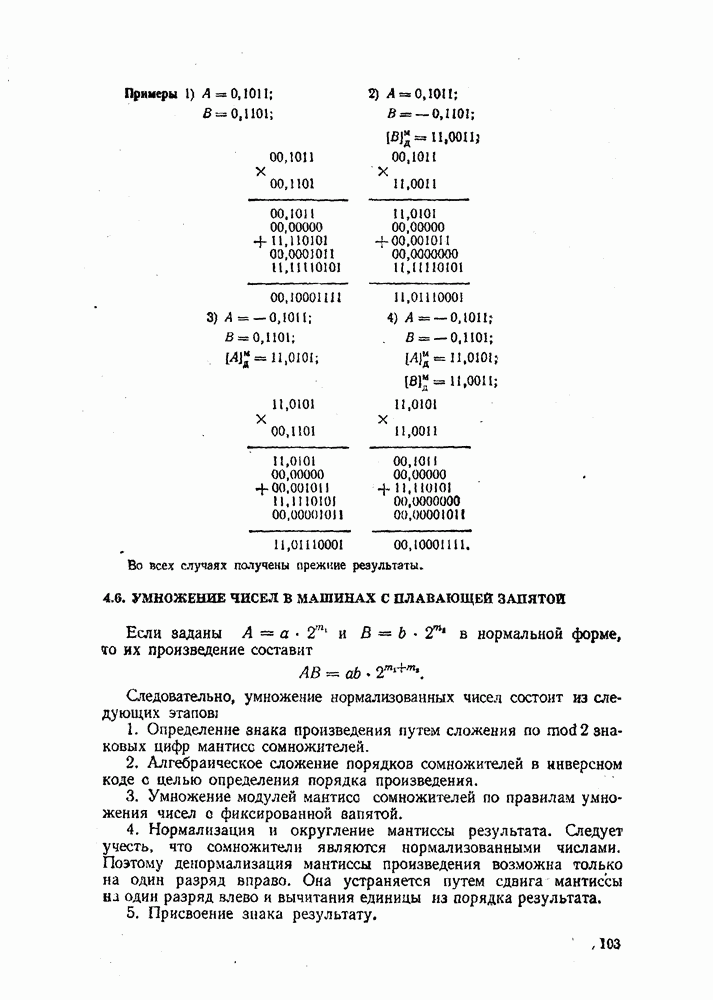
2) Алгебраическое сложение порядков сомножителей в инверсном коде с целью определения порядка произведения.

3) Умножение модулей мантисс сомножителей по правилам умножения чисел с фиксированной запятой

4) Нормализация и округление мантиссы результата. Следует учесть, что сомножители являются нормализированными числами. Поэтому денормализация мантиссы произведения возможна только на один разряд вправо. Она устраняется путем сдвига мантиссы на один разряд влево и вычитания единицы из порядка результата.

5) Присвоение знака результату.

**Примеры:**



### **1.4.** Умножение чисел с плавающей запятой

**При сравнении порядков** **возможны**пять случаев их **соотношения**:

Умножение сводится к двум простейшим действиям, рассмотренным ранее: сложению порядков сомножителей как целых чисел и перемножению мантисс как операндов с фиксированной запятой. При необходимости к ним добавляется еще нормализация мантиссы результата:

Пz = Пx + Пy

Mz = Mx \* My

Интерес здесь представляют особые случаи, которые могут возникнуть на различных этапах обработки числа.

Так при сложении чисел с фиксированной точкой (речь идёт о порядках) может возникнуть переполнение, которое сделает невозможным дальнейшую работу с этим числом, либо при перемножении мантисс сомножителей мы получим денормализованную мантиссу результата, что потребует ее коррекции с одновременной коррекцией полученного ранее порядка произведения. Все эти и некоторые другие обстоятельства необходимо учитывать при проведении данной операции.

Ряд этих случаев был рассмотрен нами при обсуждении ГОСТа IEEE 754 на представление чисел с плавающей запятой [ в лабораторной номер 4]. В то же время иногда подобные ситуации возникают в процессе выполнения самой операции, а окончательный результат оказывается допустимым числом. Рассмотрим все эти случаи.

**1.5. Экстремальные случаи.**

При выполнении операции умножения в машине с плавающей запятой может получиться переполнение отрицательного порядка, которое будет интерпретировано как машинный нуль, если программой пользователя игнорируется признак исчезновения порядка.

Может так же возникнуть и переполнение положительного порядка. В этом случае надо нормализовать мантиссу результата. Если не помогло, то формируется признак переполнения порядка.

Так как мы исходим из того, что мантиссы обоих операндов нормализованы, то есть удовлетворяют условию

1 > |Mx,y| ≥ 2-1,

то 1 > |Mz| = |Mx| \* |My| ≥ 2-2.

Таким образом, нормализация мантиссы если и потребуется, то только путем сдвига на один разряд влево. При этом порядок, естественно, следует уменьшить на 1. Отсюда вытекает следующая последовательность действий.

1. Пz = Пx + Пy
   1. Если Пz = - ∞, то Z=0 .
   2. Если Пz = + ∞, то продолжить умножение, так как последующая операция над мантиссами может привести к коррекции порядка результата в сторону его уменьшения и, тем самым, обеспечит нормальное представление числа в целом.
2. Mz = Mx \* My

Т.к. |Mx| ≥ 2-1, |My| ≥ 2-1, то |Mz| ≥ 2-2

Возможная область ненормализованной мантиссы:

2-1 > |Mz| ≥ 2-2

Если |Mz| < 2-1, то выполнить нормализацию мантиссы с одновременной коррекцией порядка:

|Mz| = |Mz| \* 2+1

Пz = Пz -1

Если в результате получим Пz = - ∞, то Z=0.

Если в ходе перемножения мантисс получим |Mz| ≥ 2-1 , но ранее при обработке порядков получили Пz = + ∞ (см п.1.2), то Z = ∞

При Z=0 выполнение программы в ЭВМ продолжается.

При Z = ∞ устанавливается флаг прерывания, и ЭВМ приостанавливает обработку данных чисел.

2. Постановка задачи

2.1. Текст задания

Эмуляция АЛУ. Реализовать операцию умножения чисел с плавающей точкой.

2.2. Примечание к заданию

Реализовать ввод двух чисел. Вычислить произведение в двоичной системе счисления в 32- битном формате, соответствующем стандарту IEEE 754. Вывести результат на экран.

3. Программная реализация

3.1. С консоли вводятся два числа в десятичной системе счисления. Затем реализуется перевод их в 32-битные контейнеры, соответствующие стандарту IEEE 754. Вызывается функция для суммы и разности введенных чисел.

3.2. Примеры

3.2.1. Тест для «a = 411.54 , b = 32.11»

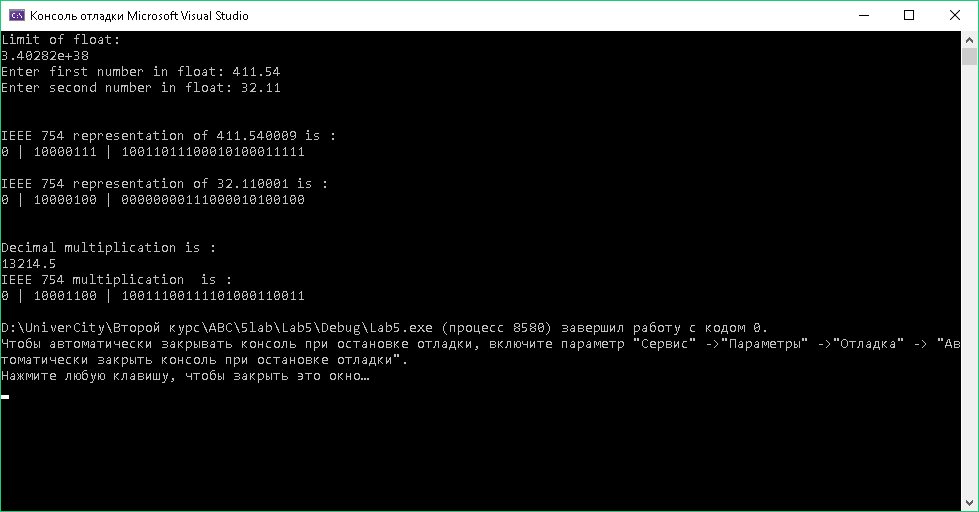


Рисунок 1. Скриншот для «a = 411.54 , b = 32.11»

Ответ: Произведение 13214.5 (в десятичной с. с.)

0 | 10001100 | 10011100111101000110011 (в двоичной)

3.2.2. Тест для «a = 132.5312, b = -52.317»

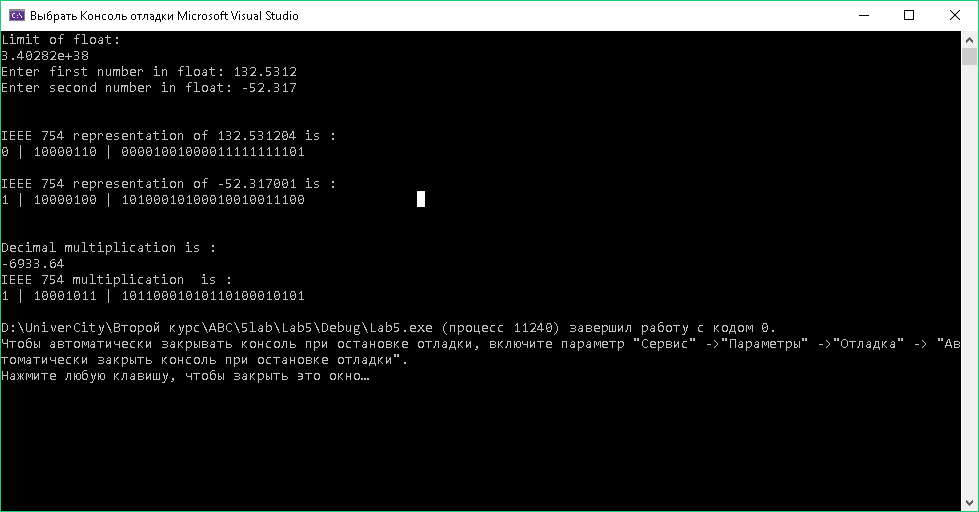


Рисунок 2. Скриншот для «a = 132.5312, b = -52.317»

Ответ: Произведение -6933.64 (в десятичной с. с.)

1 | 10001011 | 10110001010110100010101 (в двоичной)

3.2.3. Тест для «a = -53.77, b = -14.222»

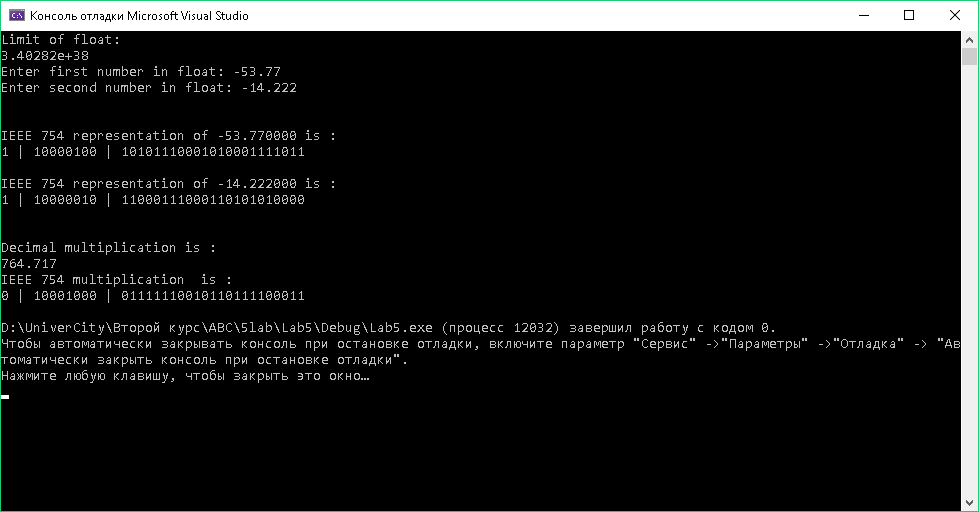


Рисунок 3. Скриншот для «a = -37.136 , b = 30.517»

Ответ: Произведение 764.717 (в десятичной с. с.)

0 | 10001000 | 01111110010110111100011 (в двоичной)

3.2.4. Тест для «a =

2000000000000000000000000, b = 2000000000000000000000000»

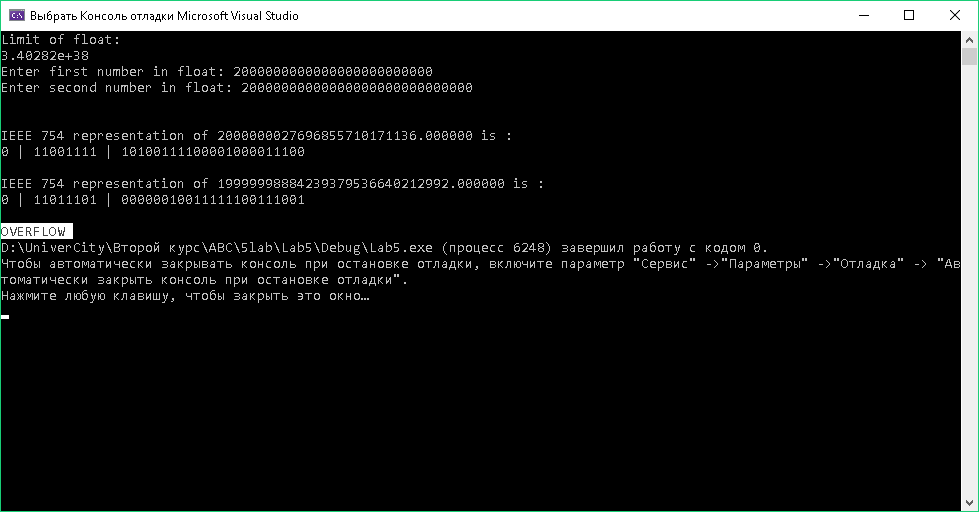


Рисунок 4. Скриншот для «a = 2000000000000000000000000, b = 2000000000000000000000000»

Ответ: OVERFLOW

3.2.7. Резюме.

В своих примерах я старался рассмотреть максимально разнообразный набор случаев. Были приведены примеры:

* Произведение двух положительных чисел
* Произведение положительного и отрицательного чисел
* Произведение двух отрицательных чисел
* Произведение, выходящее за допустимые границы.

Также были рассмотрены экстремальный случай поведения программы: переполнение результата суммы и разности.

4. Выводы

В ходе выполнения лабораторной работы ознакомился с понятиями мантиссы, экспоненты, и стандарта IEEE 754.

Так же рассмотрел работу арифметико-логического устройства (АЛУ), рассмотрел различные нюансы использования АЛУ. Для реализации использовал язык С++ и интегрированную среду разработки MVS 2019, а так же онлайн-компилятор C++ Shell(C++14).

Литература

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» /985-444-487-2 – Мн.: БГУИР, 2003. — 32 с.: ил.

2. Форум Habr. <https://habr.com/ru/post/337260/>

3. Форум Cyberforum. <https://www.cyberforum.ru/informatics/thread778486.html>

ПРИЛОЖЕНИЕ 1

Текст программы

Модуль 1. Source.cpp

#include <iostream>

#include <string>

#include <conio.h>

#include <stdio.h>

#include <cstdlib>

#include <math.h>

#include<limits>

using namespace std;

double n, bin, f1 = 0.0, f2, f3 = 0.1;

#include <stdio.h>

void printBinary(int n, int i)

{

// Prints the binary representation

// of a number n up to i-bits.

int k;

for (k = i - 1; k >= 0; k--) {

if ((n >> k) & 1)

printf("1");

else

printf("0");

}

}

typedef union {

float f;

struct

{

// Order is important.

// Here the members of the union data structure

// use the same memory (32 bits).

unsigned int mantissa : 23;

unsigned int exponent : 8;

unsigned int sign : 1;

} raw;

} myfloat;

int\* MakeArr(int\* array, myfloat var);

int\* Multiplacation(int\* arr1, int\* arr2, int\* arr3);

// Function to convert real value

void printIEEE(myfloat var)

{

// Prints the IEEE 754 representation

// of a float value (32 bits)

printf("%d | ", var.raw.sign);

printBinary(var.raw.exponent, 8);

printf(" | ");

printBinary(var.raw.mantissa, 23);

printf("\n");

}

int main()

{

myfloat var, var2, mult;

float float\_lim = fabs(numeric\_limits<float>::max());

float num, num2;

cout << "Limit of float: " << endl << float\_lim << endl;

cout << "Enter first number in float: ";

cin >> num;

cout << "Enter second number in float: ";

cin >> num2;

// Get the real value

var.f = num;

var2.f = num2;

mult.f = num \* num2;

// Get the IEEE floating point representation

printf("\n\nIEEE 754 representation of %f is : \n", var.f);

printIEEE(var);

printf("\nIEEE 754 representation of %f is : \n", var2.f);

printIEEE(var2);

int\* arr1 = new int[32];

arr1 = MakeArr(arr1, var);

int\* arr2 = new int[32];

arr2 = MakeArr(arr2, var2);

int\* arr3 = new int[32];

arr3 = Multiplacation(arr1, arr2, arr3);

if (fabs(num \* num2) >= float\_lim)

{

cout << "\nOVERFLOW";

return 0;

}

else

{

cout << "\n\nDecimal multiplication is : \n" << mult.f;

cout << "\nIEEE 754 multiplication is : \n";

printIEEE(mult);

}

return 0;

}

int\* MakeArr(int\* num, myfloat var) {

num[0] = var.raw.sign;

for (int j = 0; j < 32; j++)

num[j] = 0;

int k;

int n = var.raw.exponent;

int i = 8;

for (k = i - 1; k >= 0; k--) {

if ((n >> k) & 1)

num[8 - k] = 1;

else

num[8 - k] = 0;

}

n = var.raw.mantissa;

i = 23;

int z = 9;

for (k = i - 1; k >= 0; k--) {

if ((n >> k) & 1)

num[z] = 1;

else

num[z] = 0;

z++;

}

return num;

}

int\* Multiplacation(int\* arr1, int\* arr2, int\* sum) {

for (int i = 31; i >= 0; i--)

{

sum[i] = arr1[i] + arr2[i];

}

return sum;

}